PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT TRADEMARK OFFICE

Art Unit	: 2121	
Examiner	: Adrian L. Kennedy	
Serial No.	: 10/645,982	
Filed	: August 22, 2003	
Inventors	: Dean S. Thompson	
Title	: SYSTEM AND METHOD FOR	
	: OPTIMIZING A	
	: COMPUTER PROGRAM	Docket No.: EVO01-0001

APPEAL BRIEF

Attention: Board of Patent Appeals and Interferences
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

This Appeal Brief is being submitted in response to the Final Rejection mailed May 16, 2007. A timely Notice of Appeal was filed October 16, 2007.

This Brief is being filed within the fourth month statutory period for filing the appeal brief as set by 37 C.F.R. § 1.192(a). A Petition for Extension of Time is being filed herewith. Please charge the fee for filing the Appeal Brief and any other fee associated with this appeal to Deposit Account No. 50-2989. A copy of the claims involved in this appeal is appended hereto as an Appendix.

I. REAL PARTY IN INTEREST AND RELATED APPEALS AND INTERFERENCES

The entire interest of the subject matter of this application has been assigned to Evonics, LLC. There are no related appeals or interferences in connection with this application.

II. STATUS OF CLAIMS

Claims 1-8 and 10-20 are pending in the application. Claims 1-8 and 10-20 stand rejected. Claims 1-8 and 10-20 are on appeal.

III. STATUS OF AMENDMENTS

No amendment after final rejection has been submitted.

IV. SUMMARY OF THE CLAIMED SUBJECT MATTER

The invention of claims 1-8 and 10-19 a method for constructing at least one computer program that solves a program.

The method (method 100; *p.* 15, *ll.* 15-16; Fig(s). 1) comprises the steps of: defining a set of traits in which each trait characterizes a portion of a solution algorithm to the problem (step 112; *p.* 15, *ll.* 16-19; Fig(s). 1 and 2); defining a programming interface for at least one of the traits (step 122; *pp.* 15, *l.* 25 to *p.* 16. *l.* 4; Fig(s). 1 and 2); providing an implementation for at

least one of the defined programming interfaces (step 126; *p.* 16, *ll.* 2-4; Fig(s). 1); specifying a subtrait associated with at least one of the traits or the implementations (step 124 100; *p.* 16, *ll.* 2-10; Fig(s). 1); selecting a top-level trait that characterizes a solution to the problem (step 132; *p.* 16, *l.* 25 to *p.* 17, *l.* 18; Fig(s). 1 and 2); selecting a top-level implementation for the top-level trait (step 134; *p.* 17, *ll.* 18-21; Fig(s). 1); selecting an implementation for each subtrait required for the top-level trait or the top-level implementation (step 140; *p.* 17, *ll.* 21-27; Fig(s). 1); and recursively selecting an implementation for each subtrait associated with at least one of the traits or the implementations in order to construct a trait hierarchy that forms a computer program for solving the problem (steps 136 - 140; *p.* 17, *ll.* 21-27; Fig(s). 1).

The invention of independent claim 20 is directed to a system (method 100, problem domain-Independent components 402, and problem domain-specific components 401; *p.* 22, *l.* 21, *p.* 23, *l.* 16; Fig(s). 1 and 4) for constructing at least one computer program that solves a program. All of the following elements ion claim 20 are step plus function elements. The system comprises means for defining a set of traits in which each trait characterizes a portion of a solution algorithm to the problem (step 112, trait definitions component 408; *p.* 15 *ll.* 16-19, *p.* 24, *ll.* 4-8; Fig(s). 1, 2, and 4); means for defining a programming interface for at least one of the traits (step 122, trait support module 405; *pp.* 15, *l.* to *p.* 16. *l.* 4, *p.* 23, *ll.* 17-23; Fig(s). 1, 2, and 4); an implementation for at least one of the defined programming interfaces (step 126, trait support module 405; *p.* 16, *ll.* 2-4, *p.* 23, *ll.* 17-23; Fig(s). 1 and 4); a subtrait associated with at least one of the traits or the implementations (step 124, trait definitions component 408, trait support module 405; *p.* 16 *l* . 2-10, , *p.* 24, *ll.* 4-8, *p.* 23, *ll.* 17-23; Fig(s). 1, 2, and 4); means for selecting a top-level trait that characterizes a solution to the problem (step 132, Optimization module 406 or Interactive development environment 407; *p.* 16 *l.* 25 to *p.* 17, *l.* 18, *p.* 23, *l.* 24 to

p. 24, *l.* 3; Fig(s). 1, 2, and 4); means for selecting a top-level implementation for the top-level trait (step 134, Optimization module 406 or Interactive development environment 407; *p.* 17 *ll.* 18-21, *p.* 23, *l.* 24 to *p.* 24, *l.* 3; Fig(s). 1 and 4); means for selecting an implementation for each subtrait required for the top-level trait or the top-level implementation (step 140, Optimization module 406 or Interactive development environment 407; *p.* 17 *ll.* 21-27, *p.* 23, *l.* 24 to *p.* 24, *l.* 3; Fig(s). 1 and 4); and means for recursively selecting an implementation for each subtrait associated with at least one of the traits or the implementations in order to construct a trait hierarchy that forms a computer program for solving the problem (steps 136 – 140, evaluation module 410; *p.* 17 *ll.* 21-27, *p.* 24, *ll.* 13-18; Fig(s). 1 and 4).

Claim 2 recites that the at least one trait comprises a plurality of traits. See step 112; *p.* 15, *ll.* 16-19; Fig(s). 1 and 2.

Claim 3 recites that computer programming interface is defined for each of the traits. See step 122; *pp.* 15, *l.* 25 to *p.* 16. *l.* 4; Fig(s). 1 and 2.

Claim 4 recites that an implementation is provided for each computer programming interface. See step 126; *p.* 16, *ll.* 2-4; Fig(s). 1.

Claims 5 and 6 recite that the subtrait comprises a plurality of subtraits. See step 124 100; *p.* 16, *ll.* 2-10; Fig(s). 1.

Claims 7 and 8 recite that the top-level trait comprises a plurality of top-level traits. See step 132; *p.* 16, *l.* 25 to *p.* 17, *l.* 18; Fig(s). 1 and 2.

Claims 10 and 12 recite that the subtraits are associated with at least one of the traits, the implementation, or both. See step 124 100; *p.* 16, *ll.* 2-10; Fig(s). 1.

Claim 11 recites that the subtrait is one of the defined set of traits. See step 124 100; *p.* 16, *ll.* 2-10; Fig(s). 1.

Claim 13 further includes the features: implementing an evaluation module that executes a constructed computer program in order to determine its effectiveness in solving the problem (evaluation module 410; *p.* 17 *ll.* 21-27, *p.* 24, *ll.* 13-18; Fig(s). 1 and 4); and applying an optimization technique that carries out the steps of claim 1 to generate at least one computer program that solves the problem, and that uses feedback from the evaluation module to generate at least one additional computer program that better solves the problem (Optimization module 406, *p.* 23, *ll.* 23-25, *p.* 18, *l.* 10 to *p.* 19, *l.* 20; Fig(s). 1, 3, and 4).

Claims 14-19 recite that the optimization technique is selected from the group consisting of simulated annealing, an evolutionary algorithm, and a particle swarm optimization; allowing a user to interactively choose which trait implementations are favored or excluded at each point in each alternative computer program created by the optimization technique; defining at least one self-describing method as part of the trait implementation's interface that provides information about the trait implementation or its associated subtrait and implementing the at least one self-describing method as part of the trait implementation; using the at least one self-describing method in a user interface to provide descriptions and other detailed information about the constructed solution algorithm; using the at least one self-describing method in an optimization technique to assist in the creation of alternative computer programs; and using the at least one self-describing method in an interactive development environment to assist a user in assembling computer programs, respectively. (Optimization module 406, *p.* 23, *ll.* 23-25, *p.* 18, *l.* 10 to *p.* 19, *l.* 20; Fig(s). 1, 3, and 4)

V. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The sole issue to be considered on appeal is whether claims 1-8 and 10-20 are unpatentable under 35 U.S.C. § 103(a) in view of U.S. Pat. No. 7,047,169 (Pelikan *et al.*) (hereinafter "Pelikan"), in view of U.S. Pat. No. 6,263,325 (Yoshida *et al.*) (hereinafter "Yoshida").

The Examiner maintains that Pelikan contains each and every element recited in claims 1-8 and 9-20 with the exception of "defining a program interface" (claims 1 and 20); a "means for selecting and recursively selecting" (claim 20); a "computer programming interface" (claims 3 and 4); a "plurality of top-level traits" (claims 7 and 8); "allowing a user to interactively choose which trait implementations are favored at each point" (claim 15); "an implementation interface that provides information about a trait implementation" (claim 16); "the using of at least one self-describing method in a user interface to provide descriptions and other detailed information about the constructed solution algorithm" (claim 17); "the use of an interactive development environment" (claim 19)

The Examiner contends that it would have been obvious to one of ordinary skill in the art to remedy each of the deficiencies of Pelikan in view of Yoshida "for the purpose of solving a problem."

VI. GROUPING OF CLAIMS

Appellant submits that claims 1-8 and 10-19 are each separately patentable claims that stand or fall individually.

VII. ARGUMENT

A. The Standard For § 103 Obviousness

The conclusion that patent claims are obvious involves a preliminary determination of four factors: (1) the scope and content of the prior art; (2) the differences between the claims and the prior art; (3) the level of ordinary skill in the art; and (4) objective evidence, if any, of nonobviousness. *Uniroyal Inc. v. Rudkin-Wiley Corp.*, 837 F.2d 1044, 1050, 5 U.S.P.Q.2d 1434, 1438 (Fed. Cir. 1988), *cert. denied*, 488 U.S. 825 (1988). Each claim must be considered as a whole when weighing whether the subject matter defined by the claim would have been obvious at the time of its invention to a person of ordinary skill in the art. *In re Wright*, 848 F.2d 1216, 1219, 6 U.S.P.Q.2d 1959, 1961 (Fed. Cir. 1988).

Where, as here, references are combined to form the basis of an obviousness rejection, it must be shown that there is some teaching, suggestion, incentive or inference in the prior art that would lead a person of ordinary skill in the art to combine the relevant teachings of the art to arrive at the claimed invention. *Ex parte Levengood*, 28 U.S.P.Q. 2d 1300, 1301 (Bd.Pat.App. 1993). It is the Examiner's burden to establish a *prima facie* case of obviousness. *In re Fine*, 837 F.2d 1071, 1074, 5 U.S.P.Q. 2d 1596, 1598 (Fed. Cir. 1988); *Ex parte Levengood*, 28 U.S.P.Q.2d at 1301.

B. The Examiner Has Failed to Establish a *Prima Facie* Case of Obviousness for Any Claim on Appeal

Appellant's claimed invention is directed to a method and system for constructing one or more solutions to a problem. Pelikan and Yoshida are also concerned with the creation of one or more solutions to a problem. In particular, Pelikan is directed to a "method for optimizing a solution set for a problem" and Yoshida is directed to a "technique for obtaining an optimum solution or promising solutions to a problem according to a learning algorithm." Despite the

differing terminology employed, Appellant's claimed invention, Pelikan, and Yoshida have essentially the same top-level goal, *i.e.*, constructing one or more solutions to a problem.

Pelican and Yoshida, however, are concerned with entirely different challenges than Appellant's invention in achieving this top-level goal.

Appellant's claimed invention is concerned with the question of how to best represent candidate computer programs ("solutions") that may solve the problem by defining "traits" and "subtraits" as defined in the claims.

In contrast, Pelikan is concerned with the question of how to best find an improved "set of solutions" once a "first set of solutions" have been created. See claim 1. Yoshida is concerned with how to allow a user to be "positively involved in the execution of a learning algorithm." See *col.* 2, *l.* 54.

Like most teachings in this general area, Appellant's invention, Pelikan, and Yoshida fit into the following basic framework for automatically creating algorithms (often called "programs" or "solutions") to solve a problem:

(1) The human programmer decides how each candidate solution will be represented or encoded as a computer program;

(2) One or more initial candidate solutions are created;

(3) Each candidate solution is evaluated against the target problem;

(4) The information obtained in step (3) is used to create one or more new candidate solutions, which are generally hoped to be improvements;

(5) Step (4) is repeated until a satisfactory set of solutions is obtained.

One skilled in the art recognizes that steps (3), (4), and (5) are optional because as soon as a candidate solution has been created in step (2), the goal of creating one or more solutions to

the problem has been achieved. However, in practice steps (3), (4), and (5) are typically important to obtain a satisfactory set of solutions.

With this framework in mind, the claims of Appellant's invention can be compared to Pelican and Yoashida and it will be clear that the Examiner has completely misconstrued the teachings of Pelikan and Yoshida in setting forth the claim rejections.

Step (1): The human programmer decides how each candidate solution will be represented or encoded as a computer program.

Appellant's invention is primarily focused on this step (1) (along with step (2)). Claim 1 recites how a candidate solution can be formed by "recursively selecting an implementation for each subtrait associated with at least one of the traits or the implementations in order to construct a trait hierarchy that forms a computer program for solving the problem." Additionally, claims 1-12 and 16 are also primarily focused on this step (and step (2)).

Pelikan, in contrast, teaches that it is desirable to eliminate or minimize this step. *See col.* 2, *ll.* 11-26). In fact, Pelikan makes no specific recommendation in this area, but instead offers a laundry list of well-known approaches: "the individual members [solutions] may likewise comprise any of a number of formats, with examples including, but not limited to, k-ary strings of fixed/variable length, permutations, trees/ networks/ graphs, random keys, program codes, text, images, … and the like." See *col.* 5, *ll.* 5-16.

Yoshida does not address this step at all.

Step (2): One or more initial candidate solutions are created.

As set forth above, Appellant's invention is primarily on this step (2) (along with step (1)).

Pelikan takes no position on how this step should be performed, and instead offers a laundry list of well-known approaches: "The first set of solutions may be generated, by way of example, randomly… according to a uniform distribution, … as a result of some previous processing." *See col.* 5, *ll.* 17-24)

Yoshida does not address this step at all.

Step (3): Each candidate solution is evaluated against the target problem.

Appellant's invention does not address this step.

Pelikan teaches some specific proposals for this step: "a minimum description length metric, and the Bayesia-Direchlet metric" *See* claim 4.

Yoshida does not address this step at all.

Steps (4) and (5): The information obtained in step (3) is used to create one or more new candidate solutions, which are generally hoped to be improvements. Repeat until a satisfactory set of solutions is obtained.

These are optional steps in Appellant's invention and is not recited in the independent claims. Claim 14 recites "an optimization technique" such as "simulated annealing, an evolutionary algorithm, and a particle swarm optimization."

Pelikan's invention is primarily focused on these steps of the process, which Pelikan calls "optimizing a solution set." Claim 1, step (c) recites that this step is handled by "fitting said

second set of solutions with a probabilistic model." In addition, claims 1-3 and 5-40 are directed entirely to this step.

Yoshida's invention is also entirely directed to improving these steps by "providing a technique with which a user can be positively involved in the execution of a learning algorithm."

1. The Prior Art Does Not Disclose or Suggest Appellant's invention as Claimed

Claims 1 and 20 are directed to "constructing at least one computer program that solves a problem" by constructing an accurate and detailed design of the problem decomposition "in order to construct a trait hierarchy that forms a computer program for solving the problem" ***before an optimization method can be applied to the problem***.

To achieve this, claims 1 and 20 recite elements that are required to construct the problem decomposition, namely "defining a set of traits in which each trait characterizes a portion of a solution algorithm to the problem," "defining a programming interface for at least one of the traits," "providing an implementation for at least one of the defined programming interfaces," "specifying a subtrait associated with at least one of the traits or the implementations," "selecting a top-level trait that characterizes a solution to the problem," "selecting a top-level implementation for the top-level trait," and "selecting an implementation for each subtrait required for the top-level trait or the top-level implementation."

Pelikan, in contrast, teaches away from the need for performing these problem decomposition steps:

> As a result, traditional optimization methods application to decomposable problems has typically required accurate and detailed design of the problem decomposition before application of the method.

> High levels of effort are therefore required for solution design, adding cost and time to the solution. Further, error rates remain high when sufficient information is not available to encode the problem decomposition. These disadvantages are particularly acute when addressing problems of appreciable difficulty and/or complexity, such as hierarchically decomposable problems where dependencies, independencies, and other relationships may exist across multiple levels. For more information regarding the class of problems categorized as hierarchical, reference is made to "Sciences of the Artificial," by Herbert Simon, The MIT Press, Cambridge, Mass. (1981); herein incorporated by reference.
>
> As a result of these disadvantages, methods have been proposed to limit the need to precisely pre-code the problem decomposition. In particular, efforts have been made to develop genetic optimization methods that "learn" a problem as it is encountered through "linkage learning"--discovery of relationships between variables.

See Pelikan, *col.* 2 *ll.* 11-33.

As set forth above, Pelikan specifically teaches to minimize or eliminate this decomposition step (Steps (1) and (2) above). In fact, Pelikan makes no specific recommendation in this area. But rather states the following:

> The embodiment 100 comprises generating a first set of solutions (block 112). The solution set may comprise, by way of example, a plurality of members, with each member being a binary character string of fixed or variable length. It will be appreciated that the individual members may likewise comprise any of a number of formats, with examples including, but not limited to, k-ary strings of fixed/variable length, integer vectors of fixed/variable length, real vectors of fixed/variable length, permutations, trees/networks/graphs, random keys, program codes, text, images, production rules, logical expressions, floating point code, alphanumeric code, combinations of any of these elements, and the like.

> Further, virtually any imaginable type of individual member may be converted to a format such as a fixed/variable length n-ary string for use with an invention embodiment.
>
> ***The first set of solutions may be generated, by way of example, randomly. By way of additional examples, the first solution set may be generated according to a uniform distribution, or according to a distribution that is biased according to some expert or prior knowledge of the problem at hand. By way of still further example, the first population set may be the result of some previous processing, such as a search or optimization***. (emphasis added)

See Pelikan *col.* 5, *ll.* 1-24.

Thus, Pelikan lacks a specific teaching for each and every feature recited in independent claims 1 and 20.

The Examiner, however, latches onto a portion of a single paragraph in the Summary section of Pelikan as the basis for the rejection:

> Embodiments of the present invention are directed to methods and program products for optimizing a solution set for a problem defined over discrete variables. The iterative process of invention embodiments operates on a population of candidate solutions to the problem until termination criteria are satisfied. Embodiments of the present invention comprise steps of generating a first set of solutions, selecting a second set form the first, fitting the second set with a probabilistic model that provides for "chunking" whereby a plurality of variables may be merged into a single variable, using the model to generate a third set of solutions, and replacing at least a portion of the first set with the third set.

See Pelikan *col.* 3, *ll.* 52-63.

Clearly, Pelikan is describing an optimization method in which a "first set of solutions" are generated and then an "iterative process" is started to arrive at a target set of solutions. This would be steps (4) and (5) described above. At best, the elements of claims 1 and 20 might be part of what Pelikan calls "the first set of solutions" with the rest of this paragraph of Pelikan referring to an optimization process. Turning to claim 13, it is clear that the optimization method recited therein generates "at least one additional computer program that better solves the problem" which might be comparable to Pelikan's "third solution set" which replaces the "first solution set."

Nonetheless, Pelikan fails to describe how or what comprise this "first set of solutions." And, certainly Pelikan fails to teach or fairly suggest that the "first set of solutions" comprises "defining a set of traits in which each trait characterizes a portion of a solution algorithm to the problem," "defining a programming interface for at least one of the traits," "providing an implementation for at least one of the defined programming interfaces," "specifying a subtrait associated with at least one of the traits or the implementations," "selecting a top-level trait that characterizes a solution to the problem," "selecting a top-level implementation for the top-level trait," and "selecting an implementation for each subtrait required for the top-level trait or the top-level implementation."

Even though it appears that all of the elements recited in claims 1 and 20 would fall into the "first solution set" of Pelikan, and that Pelikan's "third solution set " replaces" the "first solution set" as part of the optimization process. The Examiner embarks on a grand adventure of redefinition in attempt to twist the terms of Pelikan's optimization method so that the Appellant's claims read on this one summary paragraph of Pelikan. Based on the Examiner's rejection of claims 1 and 20 set forth on pages 2-4 of the Final Office Action, it appears that the Examiner

has merely chosen somewhat arbitrary terms from *col.* 3, *ll.* 52-63 of Pelikan and matched them up with the terms used in Appellant's claims regardless of the reasonable meaning and functionality of those terms.

For example, claims 1 and 20 define "a set of traits," in which a trait is defined in the specification as characterizing "a portion of a solution algorithm." This feature provides components that will be assembled (through later steps) into one or more solutions to the target problem, which may be any type of problem that has an algorithmic solution. In contrast, the Examiner relies on Pelikan to teach a "first set of solutions" or "solution set for a problem." This is a set of complete candidate solutions to the target problem, where a "problem" is defined by Pelikan as being "a problem defined over discrete variables" and not a "solution algorithm" as defined by the claims. Thus, Pelikan fails to disclose this claimed element as alleged by the Examiner.

Another example, claims 1 and 20 also define "selecting an implementation for each subtrait." In contrast, the Examiner relies on Pelikan's "second set from the first" disclosure; however, this element means that a subset of the "first set of solutions" can be selected, each of which is a complete candidate solution for the target problem. Merely, selecting a first set of solutions, is not the equivalent of selecting an implementation for each subtrait since "selecting an implementation for each subtrait" requires choosing one of the available implementations for each subtrait that will become part of a complete solution to the problem. Thus, Pelikan fails to disclose this claimed element as alleged by the Examiner.

Yoshida has been relied upon by the Examiner merely to remedy the admitted deficiency of "defining a programming interface." And, in fact, Yoshida lacks a teaching of any of the remaining missing elements recited in claims 1 and 20 that the Examiner misread in Pelikan.

Yoshida, like Pelikan, is directed to an optimization method as described above not to the specific decomposition step recited in Appellant's claims.

Thus, Neither Pelikan and Yoshida, alone or in combination, disclose any of the elements defined by claim 1 or claim 20. Further, there is no teaching, suggestion, incentive or inference in either Pelikan and Yoshida that would lead a person of ordinary skill in the art to combine Pelikan and Yoshida to arrive at the claimed invention.

Even assuming the prior art teachings prompted this combination, it does not result in Appellant's claimed invention because the resulting combination lacks "defining a set of traits in which each trait characterizes a portion of a solution algorithm to the problem," "defining a programming interface for at least one of the traits," "providing an implementation for at least one of the defined programming interfaces," "specifying a subtrait associated with at least one of the traits or the implementations," "selecting a top-level trait that characterizes a solution to the problem," "selecting a top-level implementation for the top-level trait," and "selecting an implementation for each subtrait required for the top-level trait or the top-level implementation.". All of these features are present in independent claims 1 and 20.

Therefore, the Examiner has failed to make out a prima facie showing of obviousness for claims 1 and 20 on appeal because there is no teaching, suggestion, incentive or inference in the prior art that would lead a person of ordinary skill in the art to combine Pelikan and Yoshida to arrive at the claimed invention. Conversely, even if Pelikan and Yoshida are properly combinable, the combination would still not result in Appellant's claimed invention.

For the above reasons, there is no teaching, suggestion, or motivation in the prior art to combine the two cited references to come up with the claimed invention. "The mere fact that the prior art may be modified in the manner suggested by the Examiner does not make the

modification obvious unless the prior art suggested *the desirability of the modification.*" *In re* Fritch, 972 F.2d 1260, 1266, 23 U.S.P.Q.2d 1780, 1783-84 (Fed. Cir. 1992)(emphasis added). Therefore, since there is no suggestion in the prior art that would have suggested Examiner's proposed combination, the modification would not have been obvious to one having ordinary skill in the art.

Claim 2 recites that the at least one trait comprises a plurality of traits. Pelikan fails to disclose one trait as set forth above, and thus, also fails to disclose more than one trait. Yoshida does not remedy this deficiency nor has the Examiner relied on Yoshida to remedy this deficiency in Pelikan.

Claim 3 recites that a computer programming interface is defined for each of the traits. Pelikan fails to disclose a trait as set forth above, and thus, also fails to a computer programming interface for a trait. The Examiner alleges that Yoshida teaches a computer programming interface, but since neither reference teaches a trait in the first instance, it is impossible for the combination to teach a computer programming interface for a non-existent trait.

Claim 4 recites that an implementation is provided for each computer programming interface. Pelikan fails to disclose a trait as set forth above, and thus, also fails to an implementation for a computer programming interface for a trait. The Examiner alleges that Yoshida teaches an implementation, but since neither reference teaches a trait in the first instance, it is impossible for the combination to teach an implementation for a computer programming interface for a non-existent trait.

Claims 5 and 6 recite that the subtrait comprises a plurality of subtraits. Pelikan fails to disclose a subtrait as set forth above, and thus, also fails to disclose more than one subtrait.

Yoshida does not remedy this deficiency nor has the Examiner relied on Yoshida to remedy this deficiency in Pelikan.

Claims 7 and 8 recite that the top-level trait comprises a plurality of top-level traits. Pelikan fails to disclose a top-level trait as set forth above, and thus, also fails to disclose more than one top-level trait. Yoshida does not remedy this deficiency nor has the Examiner relied on Yoshida to remedy this deficiency in Pelikan.

Claims 10 and 12 recite that the subtraits are associated with at least one of the traits, the implementation, or both. Pelikan fails to disclose a subtrait as set forth above, and thus, also fails to disclose any specifics regarding that subtrait, such as the association of the subtrait. Yoshida does not remedy this deficiency nor has the Examiner relied on Yoshida to remedy this deficiency in Pelikan.

Claim 11 recites that the subtrait is one of the defined set of traits. Pelikan fails to disclose both a trait and a subtrait as set forth above, and thus, also fails to disclose the subtrait is one of the defined set of traits. Yoshida does not remedy this deficiency nor has the Examiner relied on Yoshida to remedy this deficiency in Pelikan.

Claim 13 further includes the features: implementing an evaluation module that executes a constructed computer program in order to determine its effectiveness in solving the problem and applying an optimization technique that carries out the steps of claim 1 to generate at least one computer program that solves the problem, and that uses feedback from the evaluation module to generate at least one additional computer program that better solves the problem Pelikan does appear to teach an optimization technique that replaces a first solution set with a third solution set. But, Pelikan fails to disclose that the optimization technique "executes a constructed computer program … that carries out the steps of claim 1 to generate at least one

computer program that solves the problem." Nor does that technique generate an "additional computer program that better solves the problem." There is no reasonable interpretation of "solution set" of Pelikan which is the equivalent to a computer program as claimed.

Claims 14-19 recite that the optimization technique is selected from the group consisting of simulated annealing, an evolutionary algorithm, and a particle swarm optimization; allowing a user to interactively choose which trait implementations are favored or excluded at each point in each alternative computer program created by the optimization technique; defining at least one self-describing method as part of the trait implementation's interface that provides information about the trait implementation or its associated subtrait and implementing the at least one self-describing method as part of the trait implementation; using the at least one self-describing method in a user interface to provide descriptions and other detailed information about the constructed solution algorithm; using the at least one self-describing method in an optimization technique to assist in the creation of alternative computer programs; and using the at least one self-describing method in an interactive development environment to assist a user in assembling computer programs, respectively. The examiner takes the somewhat comical position that "[b]y not specifically disclosing his method of optimization, Pelikan et al. anticipates the claiming of specific optimization algorithms in applicant's claimed invention." This conclusory statement does not discharge the examiner's burden of establishing a prima facie case of unpatentability.

Therefore, the Examiner has failed to make out a prima facie showing of obviousness for dependent claims 2-8 and 10-19 on appeal because there is no teaching, suggestion, incentive or inference in the prior art that would lead a person of ordinary skill in the art to combine Pelikan and Yoshida to arrive at the claimed invention. Conversely, even if Pelikan and Yoshida are properly combinable, the combination would still not result in Appellant's claimed invention.

VIII. CONCLUSION

For the reasons set forth above, the rejection of the claims on appeal is based on an unwarranted combination of references that does not teach or fairly suggest the subject matter of the claims. Thus, the rejection of the claims on appeal should be reversed.

Respectfully submitted,

Kenneth R. DeRosa, Reg. No. 39,549
Stuart D. Rudoler, Reg. No. 45,059
Attorney for Appellant
Rudoler & DeRosa LLC
2 Bala Plaza, Suite 300
Bala Cynwyd, PA 19004
Tel: 610-660-7753
Fax: 267-200-0796

APPENDIX

CLAIMS APPENDIX

1. (original) A method of constructing at least one computer program that solves a problem, comprising the steps of:

defining a set of traits in which each trait characterizes a portion of a solution algorithm to the problem;

defining a programming interface for at least one of the traits;

providing an implementation for at least one of the defined programming interfaces;

specifying a subtrait associated with at least one of the traits or the implementations;

selecting a top-level trait that characterizes a solution to the problem;

selecting a top-level implementation for the top-level trait;

selecting an implementation for each subtrait required for the top-level trait or the top-level implementation;

recursively selecting an implementation for each subtrait associated with at least one of the traits or the implementations in order to construct a trait hierarchy that forms a computer program for solving the problem.

2. (original) The computer program constructing method according to claim 1, wherein the at least one trait comprises a plurality of traits.

3. (original) The computer program constructing method according to claim 2, wherein a computer programming interface is defined for each of the traits.

4. (original) The computer program constructing method according to claim 3, wherein an implementation is provided for each computer programming interface.

5. (original) The computer program constructing method according to claim 1, wherein the subtrait comprises a plurality of subtraits.

6. (original) The computer program constructing method according to claim 4, wherein the subtrait comprises a plurality of subtraits.

7. (original) The computer program constructing method according to claim 1, wherein the top-level trait comprises a plurality of top-level traits.

8. (original) The computer program constructing method according to claim 6, wherein the top-level trait comprises a plurality of top-level traits.

9. (canceled)

10. (original) The computer program constructing method according to claim 5, wherein the subtraits are associated with at least one of the traits, the implementation, or both.

11. (original) The computer program constructing method according to claim 1, wherein the subtrait is one of the defined set of traits.

12. (original) The computer program constructing method according to claim 6, wherein the subtraits associated with the traits, the implementation, or both.

13. (original) The computer program constructing method according to claim 4, further comprising the steps of:

implementing an evaluation module that executes a constructed computer program in order to determine its effectiveness in solving the problem; and

applying an optimization technique that carries out the steps of claim 1 to generate at least one computer program that solves the problem, and that uses feedback from the evaluation module to generate at least one additional computer program that better solves the problem.

14. (original) The computer program constructing method according to claim 13, wherein the optimization technique is selected from the group consisting of simulated annealing, an evolutionary algorithm, and a particle swarm optimization.

15. (original) The computer program constructing method according to claim 13, further comprising the steps of: allowing a user to interactively choose which trait implementations are favored or excluded at each point in each alternative computer

program created by the optimization technique.

16. (original) The computer program constructing method according to claim 1, further comprising the steps of:

defining at least one self-describing method as part of the trait implementation's interface that provides information about the trait implementation or its associated subtrait; and

implementing the at least one self-describing method as part of the trait implementation.

17. (original) The computer program constructing method according to claim 16, further comprising the steps of:

using the at least one self-describing method in a user interface to provide descriptions and other detailed information about the constructed solution algorithm.

18. (original) The computer program constructing method according to claim 16, further comprising the steps of:

using the at least one self-describing method in an optimization technique to assist in the creation of alternative computer programs.

19. (original) The computer program constructing method according to claim 16, further comprising the steps of:

using the at least one self-describing method in an interactive development environment to assist a user in assembling computer programs.

20. (original) A system for constructing at least one computer program that solves a problem, comprising:

means for defining a set of traits in which each trait characterizes a portion of a solution algorithm to the problem;

means for defining a programming interface for at least one of the traits;

an implementation for at least one of the defined programming interfaces;

a subtrait associated with at least one of the traits or the implementations;

means for selecting a top-level trait that characterizes a solution to the problem;

means for selecting a top-level implementation for the top-level trait;

means for selecting an implementation for each subtrait required for the top-level trait or the top-level implementation;

means for recursively selecting an implementation for each subtrait associated with at least one of the traits or the implementations in order to construct a trait hierarchy that forms a computer program for solving the problem.